Ryan\_Richardson\_Final\_Projecct

Q1 2023

#Libraries:  
library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
## ✔ ggplot2 3.4.0 ✔ purrr 0.3.5   
## ✔ tibble 3.1.8 ✔ dplyr 1.0.10  
## ✔ tidyr 1.2.1 ✔ stringr 1.4.1   
## ✔ readr 2.1.3 ✔ forcats 0.5.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

library(caret)

## Loading required package: lattice  
##   
## Attaching package: 'caret'  
##   
## The following object is masked from 'package:purrr':  
##   
## lift

library(kernlab)

##   
## Attaching package: 'kernlab'  
##   
## The following object is masked from 'package:purrr':  
##   
## cross  
##   
## The following object is masked from 'package:ggplot2':  
##   
## alpha

library(randomForest)

## randomForest 4.7-1.1  
## Type rfNews() to see new features/changes/bug fixes.  
##   
## Attaching package: 'randomForest'  
##   
## The following object is masked from 'package:dplyr':  
##   
## combine  
##   
## The following object is masked from 'package:ggplot2':  
##   
## margin

library(e1071)  
library(knitr)  
library(MLmetrics)

##   
## Attaching package: 'MLmetrics'  
##   
## The following objects are masked from 'package:caret':  
##   
## MAE, RMSE  
##   
## The following object is masked from 'package:base':  
##   
## Recall

library(ggplot2)  
library(arules)

## Loading required package: Matrix  
##   
## Attaching package: 'Matrix'  
##   
## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack  
##   
##   
## Attaching package: 'arules'  
##   
## The following object is masked from 'package:kernlab':  
##   
## size  
##   
## The following object is masked from 'package:dplyr':  
##   
## recode  
##   
## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(arulesViz)

# Introduction

Poker is a popular card game with an estimated 76 Billion dollar market in 2021 [1](https://www.custommarketinsights.com/report/online-poker-market/). The goal of the game is to wager against other players on having the best combination of cards. Poker wagering includes actions such as betting, raising, calling, folding and bluffing. Bluffing is a strategy successful poker players engage in as an act of deception aimed at making your weak hand look stronger than it is with the intent of getting your opponent to fold. The goal of bluffing is to win irrespective of the hand you have been dealt. Can computer science predict and provide insights on how to improve or detect bluffing?

Previous reports have studied bluffing by reviewing body language, and facial imagery for “tells” on lying. Attempts included taking thermal images of players faces, scrapping face data from televised poker games, and sensors that detect moisture in players hands. Although these studies show promising results, the data from actually tracking the game play specifically how players are betting and reacting can provide deep insights. This data can be leveraged to provide statistics informing players of appropriate strategies.

This exercise analyzes historic poker games for bluffing trends and winning strategies by applying data science models. The data set employed is a record of the games Facebook’s AI (Pluribus)[2](Kevinwang.com) and odds data[3](betandbeat.com). This data set is unique for having all players hands available and tracking of wagering actions. Facebook’s Bot Pluribus was the first AI player to beat humans in multiplayer, Texas Holdem no limit poker. By analyzing this data, can one understand the ideal time to bluff or improve their game play? Or can you gain insights on how to detect the likelihood of whether or not someone else is bluffing and call them on their bluff? Utilizing this historical data to answer these questions can be leveraged to improve a players chance of winning.

# Data Preview and Initial Preparation

knitr::opts\_chunk$set(cache = T)  
summary(read.csv("/Users/ryanrichardson/Documents/Syr/Applied Machine Learning/Project/P-HandsX3.csv"))

## Game Winner B.Blinder BB.W   
## Length:491 Length:491 Length:491 Min. :0.0000   
## Class :character Class :character Class :character 1st Qu.:0.0000   
## Mode :character Mode :character Mode :character Median :0.0000   
## Mean :0.2892   
## 3rd Qu.:1.0000   
## Max. :1.0000   
## Bluff. Pluribus.Win. R.W.Odds Pot   
## Min. :0.0000 Min. :0.000 Min. :0.1100 Min. : 100   
## 1st Qu.:0.0000 1st Qu.:0.000 1st Qu.:0.2000 1st Qu.: 250   
## Median :0.0000 Median :0.000 Median :0.2500 Median : 500   
## Mean :0.2668 Mean :0.169 Mean :0.2594 Mean : 1455   
## 3rd Qu.:1.0000 3rd Qu.:0.000 3rd Qu.:0.2900 3rd Qu.: 1212   
## Max. :1.0000 Max. :1.000 Max. :0.5600 Max. :20950   
## PreFolds AnyBets PBetsRaises OBetsRaises   
## Min. :2.000 Min. :0.0000 Min. :0.0000 Min. :0.00   
## 1st Qu.:4.000 1st Qu.:1.0000 1st Qu.:0.0000 1st Qu.:1.00   
## Median :4.000 Median :1.0000 Median :0.0000 Median :2.00   
## Mean :4.395 Mean :0.8921 Mean :0.3116 Mean :1.65   
## 3rd Qu.:5.000 3rd Qu.:1.0000 3rd Qu.:0.0000 3rd Qu.:2.00   
## Max. :5.000 Max. :1.0000 Max. :4.0000 Max. :6.00   
## GBetsRaises MrWBetsRaises BuBetsRaises EBetsRaises   
## Min. :0.0000 Min. :0.0000 Min. :0.0000 Min. :0.0000   
## 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.0000   
## Median :0.0000 Median :0.0000 Median :0.0000 Median :0.0000   
## Mean :0.4175 Mean :0.2546 Mean :0.2749 Mean :0.3177   
## 3rd Qu.:1.0000 3rd Qu.:0.0000 3rd Qu.:0.0000 3rd Qu.:0.0000   
## Max. :5.0000 Max. :3.0000 Max. :4.0000 Max. :3.0000   
## BIBetsRaises Flop Turn River   
## Min. :0.0000 Length:491 Length:491 Length:491   
## 1st Qu.:0.0000 Class :character Class :character Class :character   
## Median :0.0000 Mode :character Mode :character Mode :character   
## Mean :0.3849   
## 3rd Qu.:1.0000   
## Max. :4.0000   
## Has.Flop Pluribus Bill Eddie   
## Min. :0.0000 Length:491 Length:491 Length:491   
## 1st Qu.:0.0000 Class :character Class :character Class :character   
## Median :1.0000 Mode :character Mode :character Mode :character   
## Mean :0.5356   
## 3rd Qu.:1.0000   
## Max. :1.0000   
## Budd Gogo MrWhite Modds   
## Length:491 Length:491 Length:491 Min. :0.1100   
## Class :character Class :character Class :character 1st Qu.:0.1500   
## Mode :character Mode :character Mode :character Median :0.1800   
## Mean :0.1908   
## 3rd Qu.:0.2200   
## Max. :0.5600   
## Godds BUodds Eodds BLodds   
## Min. :0.1100 Min. :0.1100 Min. :0.1100 Min. :0.1100   
## 1st Qu.:0.1500 1st Qu.:0.1500 1st Qu.:0.1500 1st Qu.:0.1500   
## Median :0.1800 Median :0.1800 Median :0.1800 Median :0.1900   
## Mean :0.1942 Mean :0.1911 Mean :0.1955 Mean :0.2007   
## 3rd Qu.:0.2200 3rd Qu.:0.2200 3rd Qu.:0.2200 3rd Qu.:0.2300   
## Max. :0.5600 Max. :0.5600 Max. :0.5600 Max. :0.5600   
## Podds   
## Min. :0.1100   
## 1st Qu.:0.1500   
## Median :0.1800   
## Mean :0.1944   
## 3rd Qu.:0.2200   
## Max. :0.5600

#C:/Users/beaud/Downloads/syr/IST707/P-HandsX3.csv

The dataset is parsed from a text log format and hand data cleaned up for pulling in the correct odds. Theses odds (<=.20) are used to create a bluff column for each game in order to answer questions about the data. Unwanted characters were removed. Columns that were not needed to answer the business questions were dropped. Data was also checked for any unnecessary data, irregular data or outliers. Some game IDs are duplicated but the reasoning is those games have multiple winners. Description of the fields or vectors is below.

# Data Preparation

df <- read.csv("/Users/ryanrichardson/Documents/Syr/Applied Machine Learning/Project/P-HandsX3.csv")  
#C:/Users/beaud/Downloads/syr/IST707/P-HandsX3.csv  
df <- df %>%   
 rename(  
 HasBlind = `B.Blinder`,  
 BlinderWin = `BB.W`,  
 WinnersOdds = `R.W.Odds`,  
 Bluff = `Bluff.`,  
 HasFlop = `Has.Flop`,  
 PWin = `Pluribus.Win.`,  
 BiBetsRaises = `BIBetsRaises`,  
 MrWOdds = `Modds`,  
 GOdds = `Godds`,  
 EOdds = `Eodds`,  
 POdds = `Podds`,  
 BuOdds = `BUodds`,  
 BiOdds = `BLodds`,  
 )  
df$HasBlind <- as.factor(df$HasBlind)  
df$BlinderWin <- as.factor(df$BlinderWin)  
df$Bluff <- as.factor(df$Bluff)  
df$PWin <- as.factor(df$PWin)  
df$AnyBets <- as.factor(df$AnyBets)  
df$PBetsRaises <- as.factor(df$PBetsRaises)  
df$PBetsRaises <- ordered(df$PBetsRaises, levels = c(0,1,2,3,4))  
df$OBetsRaises <- as.factor(df$OBetsRaises)  
df$OBetsRaises <- ordered(df$OBetsRaises, levels = c(0,1,2,3,4,5,6))  
df$GBetsRaises <- as.factor(df$GBetsRaises)  
df$GBetsRaises <- ordered(df$GBetsRaises, levels = c(0,1,2,3,4,5))  
df$MrWBetsRaises <- as.factor(df$MrWBetsRaises)  
df$MrWBetsRaises <- ordered(df$MrWBetsRaises, levels = c(0,1,2,3))  
df$BuBetsRaises <- as.factor(df$BuBetsRaises)  
df$BuBetsRaises <- ordered(df$BuBetsRaises, levels = c(0,1,2,3,4))  
df$EBetsRaises <- as.factor(df$EBetsRaises)  
df$EBetsRaises <- ordered(df$EBetsRaises, levels = c(0,1,2,3))  
df$BiBetsRaises <- as.factor(df$BiBetsRaises)  
df$BiBetsRaises <- ordered(df$BiBetsRaises, levels = c(0,1,2,3,4))  
df$HasFlop <- as.factor(df$HasFlop)  
df <- df %>% mutate\_if(is.character,as.factor)  
  
  
summary(df)

## Game Winner HasBlind BlinderWin Bluff PWin   
## #31052 : 2 Bill :93 Bill :82 0:349 0:360 0:408   
## #32023 : 2 Budd :69 Budd :83 1:142 1:131 1: 83   
## #33021 : 2 Eddie :83 Eddie :81   
## #30000 : 1 Gogo :93 Gogo :84   
## #30001 : 1 MrWhite :70 MrWhite :80   
## #30002 : 1 Pluribus:83 Pluribus:81   
## (Other):482   
## WinnersOdds Pot PreFolds AnyBets PBetsRaises  
## Min. :0.1100 Min. : 100 Min. :2.000 0: 53 0:392   
## 1st Qu.:0.2000 1st Qu.: 250 1st Qu.:4.000 1:438 1: 56   
## Median :0.2500 Median : 500 Median :4.000 2: 33   
## Mean :0.2594 Mean : 1455 Mean :4.395 3: 9   
## 3rd Qu.:0.2900 3rd Qu.: 1212 3rd Qu.:5.000 4: 1   
## Max. :0.5600 Max. :20950 Max. :5.000   
##   
## OBetsRaises GBetsRaises MrWBetsRaises BuBetsRaises EBetsRaises BiBetsRaises  
## 0: 96 0:362 0:400 0:402 0:387 0:363   
## 1:144 1: 69 1: 66 1: 53 1: 58 1: 79   
## 2:142 2: 49 2: 16 2: 27 2: 40 2: 38   
## 3: 66 3: 7 3: 9 3: 8 3: 6 3: 10   
## 4: 32 4: 3 4: 1 4: 1   
## 5: 10 5: 1   
## 6: 1   
## Flop Turn River HasFlop Pluribus   
## :228 :303 :354 0:228 5A : 7   
## [3c 8h 2s]: 2 [Ks] : 9 [Ks] : 7 1:263 J5 : 6   
## [Td 9s 9c]: 2 [9h] : 8 [Ah] : 6 3J : 5   
## [Th 5h 8c]: 2 [5c] : 7 [Ts] : 6 4T : 5   
## [2c 3d 7c]: 1 [Js] : 7 [6c] : 5 5Q : 5   
## [2c 4d Kd]: 1 [Qd] : 7 [8s] : 5 7T : 5   
## (Other) :255 (Other):150 (Other):108 (Other):458   
## Bill Eddie Budd Gogo MrWhite   
## 93 : 7 22 : 6 93 : 8 2Q : 8 3K : 7   
## KK : 7 28 : 6 37 : 6 77 : 7 77 : 6   
## A3 : 6 K9 : 6 4K : 6 56 : 6 94 : 6   
## K9 : 6 Q6 : 6 63 : 6 A5 : 6 24 : 5   
## 52 : 5 T7 : 6 J9 : 6 KJ : 6 39 : 5   
## 54 : 5 25 : 5 68 : 5 39 : 5 76 : 5   
## (Other):455 (Other):456 (Other):454 (Other):453 (Other):457   
## MrWOdds GOdds BuOdds EOdds   
## Min. :0.1100 Min. :0.1100 Min. :0.1100 Min. :0.1100   
## 1st Qu.:0.1500 1st Qu.:0.1500 1st Qu.:0.1500 1st Qu.:0.1500   
## Median :0.1800 Median :0.1800 Median :0.1800 Median :0.1800   
## Mean :0.1908 Mean :0.1942 Mean :0.1911 Mean :0.1955   
## 3rd Qu.:0.2200 3rd Qu.:0.2200 3rd Qu.:0.2200 3rd Qu.:0.2200   
## Max. :0.5600 Max. :0.5600 Max. :0.5600 Max. :0.5600   
##   
## BiOdds POdds   
## Min. :0.1100 Min. :0.1100   
## 1st Qu.:0.1500 1st Qu.:0.1500   
## Median :0.1900 Median :0.1800   
## Mean :0.2007 Mean :0.1944   
## 3rd Qu.:0.2300 3rd Qu.:0.2200   
## Max. :0.5600 Max. :0.5600   
##

PBluff <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$POdds <= .2 & df[i,]$PBetsRaises != 0) {  
 PBluff <- append(PBluff, 1)  
 } else {  
 PBluff <- append(PBluff, 0)  
 }  
}  
df$PBluff <- as.factor(PBluff)

GBluff <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$GOdds <= .2 & df[i,]$GBetsRaises != 0) {  
 GBluff <- append(GBluff, 1)  
 } else {  
 GBluff <- append(GBluff, 0)  
 }  
}  
df$GBluff <- as.factor(GBluff)

MrWBluff <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$MrWOdds <= .2 & df[i,]$MrWBetsRaises != 0) {  
 MrWBluff <- append(MrWBluff, 1)  
 } else {  
 MrWBluff <- append(MrWBluff, 0)  
 }  
}  
df$MrWBluff <- as.factor(MrWBluff)

BuBluff <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$BuOdds <= .2 & df[i,]$BuBetsRaises != 0) {  
 BuBluff <- append(BuBluff, 1)  
 } else {  
 BuBluff <- append(BuBluff, 0)  
 }  
}  
df$BuBluff <- as.factor(BuBluff)

EBluff <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$EOdds <= .2 & df[i,]$EBetsRaises != 0) {  
 EBluff <- append(EBluff, 1)  
 } else {  
 EBluff <- append(EBluff, 0)  
 }  
}  
df$EBluff <- as.factor(EBluff)

BiBluff <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$BiOdds <= .2 & df[i,]$BiBetsRaises != 0) {  
 BiBluff <- append(BiBluff, 1)  
 } else {  
 BiBluff <- append(BiBluff, 0)  
 }  
}  
df$BiBluff <- as.factor(BiBluff)

PBluffWin <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$PBluff == 1 & df[i,]$Winner == 'Pluribus') {  
 PBluffWin <- append(PBluffWin, 1)  
 } else {  
 PBluffWin <- append(PBluffWin, 0)  
 }  
}  
df$PBluffWin <- as.factor(PBluffWin)

GBluffWin <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$GBluff == 1 & df[i,]$Winner == 'Gogo') {  
 GBluffWin <- append(GBluffWin, 1)  
 } else {  
 GBluffWin <- append(GBluffWin, 0)  
 }  
}  
df$GBluffWin <- as.factor(GBluffWin)

MrWBluffWin <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$MrWBluff == 1 & df[i,]$Winner == 'MrWhite') {  
 MrWBluffWin <- append(MrWBluffWin, 1)  
 } else {  
 MrWBluffWin <- append(MrWBluffWin, 0)  
 }  
}  
df$MrWBluffWin <- as.factor(MrWBluffWin)

BuBluffWin <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$BuBluff == 1 & df[i,]$Winner == 'Budd') {  
 BuBluffWin <- append(BuBluffWin, 1)  
 } else {  
 BuBluffWin <- append(BuBluffWin, 0)  
 }  
}  
df$BuBluffWin <- as.factor(BuBluffWin)

EBluffWin <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$EBluff == 1 & df[i,]$Winner == 'Eddie') {  
 EBluffWin <- append(EBluffWin, 1)  
 } else {  
 EBluffWin <- append(EBluffWin, 0)  
 }  
}  
df$EBluffWin <- as.factor(EBluffWin)

BiBluffWin <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$BiBluff == 1 & df[i,]$Winner == 'Bill') {  
 BiBluffWin <- append(BiBluffWin, 1)  
 } else {  
 BiBluffWin <- append(BiBluffWin, 0)  
 }  
}  
df$BiBluffWin <- as.factor(BiBluffWin)

AnyBluffWin <- c()  
  
zeros <- rep(0, 6)  
  
for (i in 1:nrow(df)) {  
 if (all(df[i,40:45] == zeros)) {  
 AnyBluffWin <- append(AnyBluffWin, 0)  
 } else {  
 AnyBluffWin <- append(AnyBluffWin, 1)  
 }  
}  
df$AnyBluffWin <- as.factor(AnyBluffWin)

PBluffLoss <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$PBluff == 1 & df[i,]$Winner != 'Pluribus') {  
 PBluffLoss <- append(PBluffLoss, 1)  
 } else {  
 PBluffLoss <- append(PBluffLoss, 0)  
 }  
}  
df$PBluffLoss <- as.factor(PBluffLoss)

GBluffLoss <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$GBluff == 1 & df[i,]$Winner != 'Gogo') {  
 GBluffLoss <- append(GBluffLoss, 1)  
 } else {  
 GBluffLoss <- append(GBluffLoss, 0)  
 }  
}  
df$GBluffLoss <- as.factor(GBluffLoss)

MrWBluffLoss <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$MrWBluff == 1 & df[i,]$Winner != 'MrWhite') {  
 MrWBluffLoss <- append(MrWBluffLoss, 1)  
 } else {  
 MrWBluffLoss <- append(MrWBluffLoss, 0)  
 }  
}  
df$MrWBluffLoss <- as.factor(MrWBluffLoss)

BuBluffLoss <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$BuBluff == 1 & df[i,]$Winner != 'Budd') {  
 BuBluffLoss <- append(BuBluffLoss, 1)  
 } else {  
 BuBluffLoss <- append(BuBluffLoss, 0)  
 }  
}  
df$BuBluffLoss <- as.factor(BuBluffLoss)

EBluffLoss <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$EBluff == 1 & df[i,]$Winner != 'Eddie') {  
 EBluffLoss <- append(EBluffLoss, 1)  
 } else {  
 EBluffLoss <- append(EBluffLoss, 0)  
 }  
}  
df$EBluffLoss <- as.factor(EBluffLoss)

BiBluffLoss <- c()  
for(i in 1:nrow(df)) {  
 if(df[i,]$BiBluff == 1 & df[i,]$Winner != 'Bill') {  
 BiBluffLoss <- append(BiBluffLoss, 1)  
 } else {  
 BiBluffLoss <- append(BiBluffLoss, 0)  
 }  
}  
df$BiBluffLoss <- as.factor(BiBluffLoss)

AnyBluffLoss <- c()  
  
for (i in 1:nrow(df)) {  
 if (all(df[i,47:52] == zeros)) {  
 AnyBluffLoss <- append(AnyBluffLoss, 0)  
 } else {  
 AnyBluffLoss <- append(AnyBluffLoss, 1)  
 }  
}  
df$AnyBluffLoss <- as.factor(AnyBluffLoss)

df <- df %>%   
 group\_by(Winner) %>%   
 mutate(CumulativeEarnings = cumsum(Pot))

print(head(df))

## # A tibble: 6 × 54  
## # Groups: Winner [5]  
## Game Winner HasBl…¹ Blind…² Bluff PWin Winne…³ Pot PreFo…⁴ AnyBets PBets…⁵  
## <fct> <fct> <fct> <fct> <fct> <fct> <dbl> <dbl> <int> <fct> <ord>   
## 1 #300… Bill Gogo 0 0 0 0.35 250 5 1 0   
## 2 #300… Gogo Budd 0 1 0 0.2 200 5 1 0   
## 3 #300… Eddie Eddie 1 1 0 0.18 450 4 1 0   
## 4 #300… Budd Bill 0 0 0 0.32 4250 4 1 0   
## 5 #300… Eddie Plurib… 0 0 0 0.27 600 5 1 0   
## 6 #300… Pluri… MrWhite 0 0 1 0.4 500 5 1 1   
## # … with 43 more variables: OBetsRaises <ord>, GBetsRaises <ord>,  
## # MrWBetsRaises <ord>, BuBetsRaises <ord>, EBetsRaises <ord>,  
## # BiBetsRaises <ord>, Flop <fct>, Turn <fct>, River <fct>, HasFlop <fct>,  
## # Pluribus <fct>, Bill <fct>, Eddie <fct>, Budd <fct>, Gogo <fct>,  
## # MrWhite <fct>, MrWOdds <dbl>, GOdds <dbl>, BuOdds <dbl>, EOdds <dbl>,  
## # BiOdds <dbl>, POdds <dbl>, PBluff <fct>, GBluff <fct>, MrWBluff <fct>,  
## # BuBluff <fct>, EBluff <fct>, BiBluff <fct>, PBluffWin <fct>, …

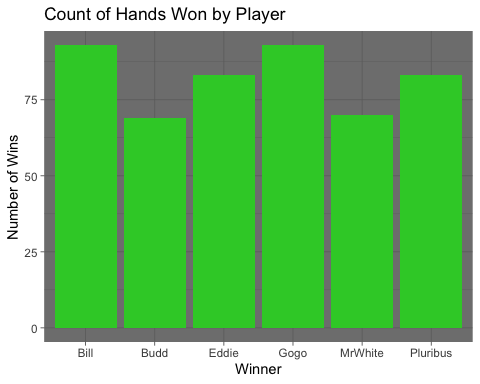
print(summary(df))

## Game Winner HasBlind BlinderWin Bluff PWin   
## #31052 : 2 Bill :93 Bill :82 0:349 0:360 0:408   
## #32023 : 2 Budd :69 Budd :83 1:142 1:131 1: 83   
## #33021 : 2 Eddie :83 Eddie :81   
## #30000 : 1 Gogo :93 Gogo :84   
## #30001 : 1 MrWhite :70 MrWhite :80   
## #30002 : 1 Pluribus:83 Pluribus:81   
## (Other):482   
## WinnersOdds Pot PreFolds AnyBets PBetsRaises  
## Min. :0.1100 Min. : 100 Min. :2.000 0: 53 0:392   
## 1st Qu.:0.2000 1st Qu.: 250 1st Qu.:4.000 1:438 1: 56   
## Median :0.2500 Median : 500 Median :4.000 2: 33   
## Mean :0.2594 Mean : 1455 Mean :4.395 3: 9   
## 3rd Qu.:0.2900 3rd Qu.: 1212 3rd Qu.:5.000 4: 1   
## Max. :0.5600 Max. :20950 Max. :5.000   
##   
## OBetsRaises GBetsRaises MrWBetsRaises BuBetsRaises EBetsRaises BiBetsRaises  
## 0: 96 0:362 0:400 0:402 0:387 0:363   
## 1:144 1: 69 1: 66 1: 53 1: 58 1: 79   
## 2:142 2: 49 2: 16 2: 27 2: 40 2: 38   
## 3: 66 3: 7 3: 9 3: 8 3: 6 3: 10   
## 4: 32 4: 3 4: 1 4: 1   
## 5: 10 5: 1   
## 6: 1   
## Flop Turn River HasFlop Pluribus   
## :228 :303 :354 0:228 5A : 7   
## [3c 8h 2s]: 2 [Ks] : 9 [Ks] : 7 1:263 J5 : 6   
## [Td 9s 9c]: 2 [9h] : 8 [Ah] : 6 3J : 5   
## [Th 5h 8c]: 2 [5c] : 7 [Ts] : 6 4T : 5   
## [2c 3d 7c]: 1 [Js] : 7 [6c] : 5 5Q : 5   
## [2c 4d Kd]: 1 [Qd] : 7 [8s] : 5 7T : 5   
## (Other) :255 (Other):150 (Other):108 (Other):458   
## Bill Eddie Budd Gogo MrWhite   
## 93 : 7 22 : 6 93 : 8 2Q : 8 3K : 7   
## KK : 7 28 : 6 37 : 6 77 : 7 77 : 6   
## A3 : 6 K9 : 6 4K : 6 56 : 6 94 : 6   
## K9 : 6 Q6 : 6 63 : 6 A5 : 6 24 : 5   
## 52 : 5 T7 : 6 J9 : 6 KJ : 6 39 : 5   
## 54 : 5 25 : 5 68 : 5 39 : 5 76 : 5   
## (Other):455 (Other):456 (Other):454 (Other):453 (Other):457   
## MrWOdds GOdds BuOdds EOdds   
## Min. :0.1100 Min. :0.1100 Min. :0.1100 Min. :0.1100   
## 1st Qu.:0.1500 1st Qu.:0.1500 1st Qu.:0.1500 1st Qu.:0.1500   
## Median :0.1800 Median :0.1800 Median :0.1800 Median :0.1800   
## Mean :0.1908 Mean :0.1942 Mean :0.1911 Mean :0.1955   
## 3rd Qu.:0.2200 3rd Qu.:0.2200 3rd Qu.:0.2200 3rd Qu.:0.2200   
## Max. :0.5600 Max. :0.5600 Max. :0.5600 Max. :0.5600   
##   
## BiOdds POdds PBluff GBluff MrWBluff BuBluff EBluff   
## Min. :0.1100 Min. :0.1100 0:473 0:457 0:470 0:476 0:474   
## 1st Qu.:0.1500 1st Qu.:0.1500 1: 18 1: 34 1: 21 1: 15 1: 17   
## Median :0.1900 Median :0.1800   
## Mean :0.2007 Mean :0.1944   
## 3rd Qu.:0.2300 3rd Qu.:0.2200   
## Max. :0.5600 Max. :0.5600   
##   
## BiBluff PBluffWin GBluffWin MrWBluffWin BuBluffWin EBluffWin BiBluffWin  
## 0:464 0:476 0:469 0:480 0:483 0:482 0:469   
## 1: 27 1: 15 1: 22 1: 11 1: 8 1: 9 1: 22   
##   
##   
##   
##   
##   
## AnyBluffWin PBluffLoss GBluffLoss MrWBluffLoss BuBluffLoss EBluffLoss  
## 0:404 0:488 0:479 0:481 0:484 0:483   
## 1: 87 1: 3 1: 12 1: 10 1: 7 1: 8   
##   
##   
##   
##   
##   
## BiBluffLoss AnyBluffLoss CumulativeEarnings  
## 0:486 0:446 Min. : 200   
## 1: 5 1: 45 1st Qu.: 30250   
## Median : 61998   
## Mean : 63627   
## 3rd Qu.: 90092   
## Max. :160750   
##

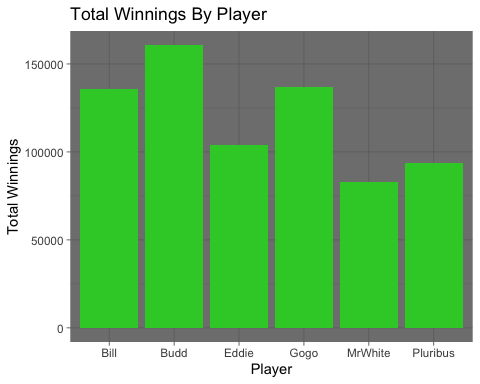
In order to prepare the data, all string columns were converted to factors to potentially identify patterns and enable potential analysis by different types of hands. Boolean columns were also converted to factors to enable functionality with certain algorithms that will be used later. Additionally, calculated columns were created to identify who bluffed during particularly games and whether or not that bluff would lead to the player winning the hand. Similarly, lost bluffs were also added which will enable analysis on players that are not as good at bluffing. Finally, a new column of cumulative winnings for each player across hands was also added. With this preparation, some high level observations can be identified. Generally, all players, including Pluribus, are conservative with their bluffs. Interestingly, three game ID’s are duplicated - after review, this is due to the how the data stores hands where the pot is split. The data is stored in multiple rows with the same game id but the outcome (ie, the pot payout) is different to capture how it is paid to multiple players. It is now appropriate to start visualizing the data for Exploratory Data Analysis.

# Exploratory Data Analysis

winCount <- df %>% count(Winner)  
ggplot(winCount, aes(x = Winner, y = n)) + geom\_col(fill = 'limegreen') + labs(x = 'Winner', y = 'Number of Wins', title = "Count of Hands Won by Player") + theme\_dark()

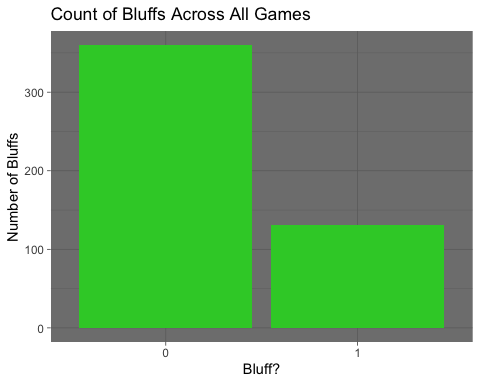
 Budd and MrWhite notably lag the rest of the group in wins, while Eddie and Pluribus stand in the middle. Bill and Gogo win the most hands. That said, number of hands won doesn’t necessarily indicate who actually wins the game:

potWinnings <- aggregate(Pot ~ Winner, data = df, FUN = function(x) sum(x))  
ggplot(potWinnings, aes(x = Winner, y = Pot)) + geom\_col(fill = 'limegreen') + labs(x = "Player", y = 'Total Winnings', title = "Total Winnings By Player") + theme\_dark()



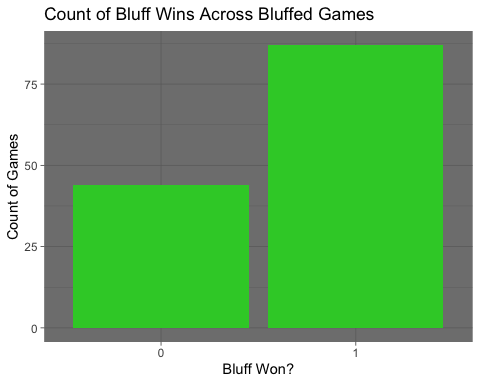
Even though Budd does not win as many hands, the hands he wins really matter. Bill and Gogo remain solid performers while Eddie and MrWhite still lag. Pluribus has the worst return on hands won, only outperforming MrWhite for total earnings.

bluffCount <- df %>% count(Bluff)  
ggplot(bluffCount, aes(x = Bluff, y = n)) + geom\_col(fill = 'limegreen') + labs(x = 'Bluff?', y = 'Number of Bluffs', title = "Count of Bluffs Across All Games") + theme\_dark()

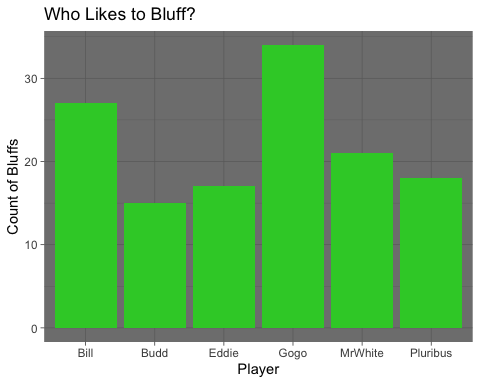


Across all of the hands, relatively few games involved someone bluffing.

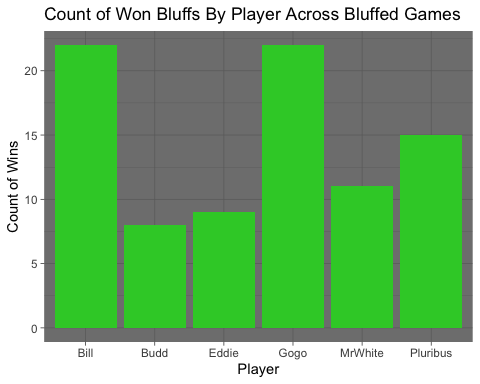
bluffedGames <- df %>% filter(Bluff == 1)  
bluffWinCount <- bluffedGames %>% count(AnyBluffWin)  
ggplot(bluffWinCount, aes(x = AnyBluffWin, y = n)) + geom\_col(fill = 'limegreen') + labs(x = 'Bluff Won?', y = 'Count of Games', title = "Count of Bluff Wins Across Bluffed Games") + theme\_dark()

 However, even though the players were conservative in their bluffs, they tended to win when they did bluff.

bluffCountsByPlayer <- gather(df[,34:39], key = "Player", value = "Bluffs")  
  
aggBluffCounts <- aggregate(Bluffs ~ Player, data = bluffCountsByPlayer, FUN = function(x) sum(as.numeric(x == 1)))  
  
nameList <- c('Bill', 'Budd', 'Eddie', 'Gogo', 'MrWhite', 'Pluribus')  
  
aggBluffCounts$Player <- nameList  
  
ggplot(aggBluffCounts, aes(x = Player, y = Bluffs)) + geom\_col(fill = 'limegreen') + labs(x = "Player", y = 'Count of Bluffs', title = "Who Likes to Bluff?") + theme\_dark()

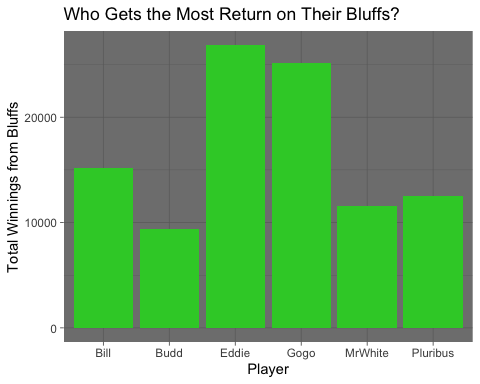


bluffWinnerList <- gather(df[,40:45], key = "Player", value = "nWon")  
  
bluffWinnerCount <- aggregate(nWon ~ Player, data = bluffWinnerList, FUN = function(x) sum(as.numeric(x == 1)))  
  
bluffWinnerCount$Player <- nameList  
  
ggplot(bluffWinnerCount, aes(x = Player, y = nWon)) + geom\_col(fill = 'limegreen') + labs(x = "Player", y = 'Count of Wins', title = "Count of Won Bluffs By Player Across Bluffed Games") + theme\_dark()

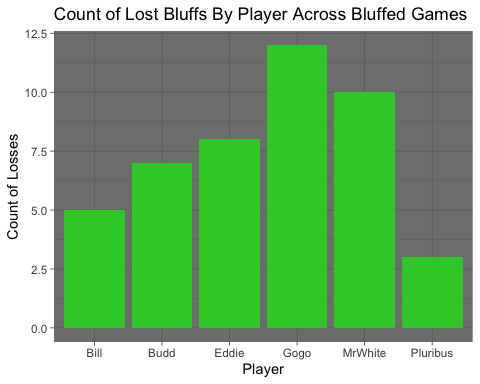


Gogo and Bill won the most bluffs by a fair margin. The other three players lagged substantially, though Pluribus and MrWhite bluffed slightly more often and calling it a uniform distribution would be a stretch. As observed above however, hands won does not necessarily indicate who performs the best on their bluffing:

bluffPotWinnings <- aggregate(Pot ~ Winner, data = bluffedGames, FUN = function(x) sum(x))  
ggplot(bluffPotWinnings, aes(x = Winner, y = Pot)) + geom\_col(fill = 'limegreen') + labs(x = "Player", y = 'Total Winnings from Bluffs', title = "Who Gets the Most Return on Their Bluffs?") + theme\_dark()

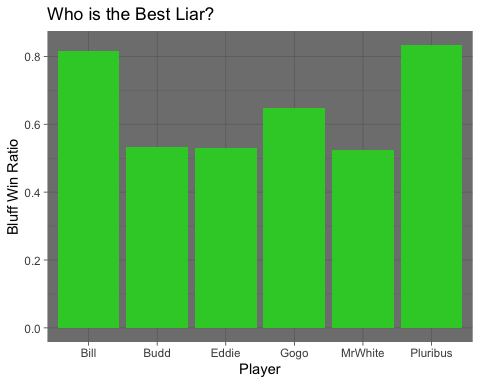
 Here, it can be observed that Eddie excels at getting good return when he does bluff, even though he is one of the rarer bluffers. Gogo also gets excellent return for her bluffs, which might be expected given her bluff frequency compared to the other players. Bill does not do nearly as well even though he bluffs almost as often as Gogo. Bluff performance can also be thought of in contrast to its negative:

bluffLoserList <- gather(df[,47:52], key = "Player", value = "nLost")  
  
bluffLoserCount <- aggregate(nLost ~ Player, data = bluffLoserList, FUN = function(x) sum(as.numeric(x == 1)))  
  
bluffLoserCount$Player <- nameList  
  
ggplot(bluffLoserCount, aes(x = Player, y = nLost)) + geom\_col(fill = 'limegreen') + labs(x = "Player", y = 'Count of Losses', title = "Count of Lost Bluffs By Player Across Bluffed Games") + theme\_dark()



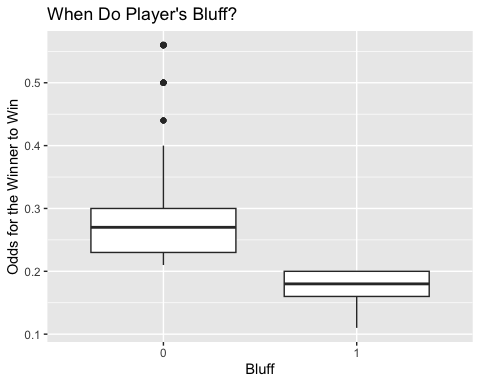
Even though Gogo bluffs a lot and wins alot, she also loses the most bluffs. MrWhite’s bluff performance by every metric so far remains dismal. Pluribus and Bill rarely lose their bluffs. This can also be visualized as a ratio:

bluffRatio <- inner\_join(bluffWinnerCount, bluffLoserCount, by = "Player")  
  
# calculate win ratio  
bluffRatio <- bluffRatio %>%   
 mutate(total = nWon + nLost,  
 win\_ratio = nWon / total)  
  
# create bar chart  
ggplot(bluffRatio, aes(x = Player, y = win\_ratio)) + geom\_bar(stat = "identity", fill = 'limegreen') + ggtitle("Who is the Best Liar?") + ylab("Bluff Win Ratio") + xlab("Player") + theme\_dark()



Budd, Eddie, and Gogo are relatively uniform in their bluff win ratio. MrWhite’s performance continues to appear lackluster. Bill and Pluribus have exceptionally strong ratios, winning more than 80% of their bluffs.

ggplot(df, aes(x = Bluff, y = WinnersOdds)) + geom\_boxplot() +   
 labs(x = 'Bluff', y = "Odds for the Winner to Win", title = "When Do Player's Bluff?")



It should be noted that this is a slightly misleading plot. There is a hard cutoff for bluffs at 20% odds because anything greater than 20% was not mathematically considered a bluff. This naturally explains the higher odds that cannot be seen in bluffed hands. Additionally, odds for winning do not drop below 10%. That said, it is interesting to note that the median bluff appears to be around 18% odds or so. Players appear to prefer bluffing when there is still some path to winning.

# Correlation Analysis

Here with correlation the exercise looks at what vectors have positive and negative correlation to bluffing. A correlation matrix could be useful for players wanting to know what actions lean towards or against bluffing.

#New df for alternate workflow  
df2 <- read\_csv("/Users/ryanrichardson/Documents/Syr/Applied Machine Learning/Project/P-HandsX3.csv")

## Rows: 491 Columns: 33  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (12): Game, Winner, B.Blinder, Flop, Turn, River, Pluribus, Bill, Eddie,...  
## dbl (21): BB=W, Bluff?, Pluribus Win?, R.W.Odds, Pot, PreFolds, AnyBets, PBe...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

bluffSet <- data.frame(bluff=df2$`Bluff?`,  
 BBwin=scale(df2$`BB=W`),  
 preFolds=scale(df2$PreFolds),  
 Pot=scale(df2$Pot),   
 Bets=(scale(df2$PBetsRaises + df2$OBetsRaises)),   
 Flop=scale(df2$`Has Flop`),  
 PBR=scale(df2$PBetsRaises),  
 GBR=scale(df2$GBetsRaises),  
 MrWBR=scale(df2$MrWBetsRaises),  
 BuBR=scale(df2$BuBetsRaises),  
 EBR=scale(df2$EBetsRaises),  
 BIBR=scale(df2$BIBetsRaises),  
 Modds=df2$Modds,  
 Godds=df2$Godds,  
 BUodds=df2$BUodds,  
 Eodds=df2$Eodds,  
 BLodds=df2$BLodds,  
 Podds=df2$Podds  
 ) #add flop details?  
  
res <- cor(bluffSet)  
round(res, 2)

## bluff BBwin preFolds Pot Bets Flop PBR GBR MrWBR BuBR EBR  
## bluff 1.00 0.43 0.10 -0.13 -0.24 -0.10 0.03 -0.07 -0.09 -0.15 -0.08  
## BBwin 0.43 1.00 0.07 -0.08 -0.16 -0.05 -0.07 -0.04 -0.08 -0.06 -0.03  
## preFolds 0.10 0.07 1.00 -0.30 -0.60 -0.90 -0.19 -0.22 -0.10 -0.14 -0.27  
## Pot -0.13 -0.08 -0.30 1.00 0.63 0.34 0.18 0.27 0.11 0.26 0.03  
## Bets -0.24 -0.16 -0.60 0.63 1.00 0.68 0.27 0.43 0.22 0.29 0.23  
## Flop -0.10 -0.05 -0.90 0.34 0.68 1.00 0.21 0.26 0.12 0.18 0.26  
## PBR 0.03 -0.07 -0.19 0.18 0.27 0.21 1.00 -0.12 -0.14 -0.12 -0.03  
## GBR -0.07 -0.04 -0.22 0.27 0.43 0.26 -0.12 1.00 -0.03 -0.04 -0.12  
## MrWBR -0.09 -0.08 -0.10 0.11 0.22 0.12 -0.14 -0.03 1.00 -0.11 -0.10  
## BuBR -0.15 -0.06 -0.14 0.26 0.29 0.18 -0.12 -0.04 -0.11 1.00 -0.12  
## EBR -0.08 -0.03 -0.27 0.03 0.23 0.26 -0.03 -0.12 -0.10 -0.12 1.00  
## BIBR -0.09 -0.01 -0.14 0.26 0.32 0.19 -0.10 -0.08 -0.10 -0.04 -0.17  
## Modds -0.19 -0.13 -0.17 0.05 0.15 0.11 -0.11 -0.02 0.53 -0.04 0.05  
## Godds -0.18 -0.15 -0.13 0.17 0.27 0.16 -0.07 0.58 0.08 -0.04 -0.07  
## BUodds -0.20 -0.12 -0.11 0.27 0.22 0.11 -0.06 0.01 -0.04 0.62 -0.09  
## Eodds -0.17 -0.03 -0.18 0.00 0.18 0.16 -0.09 0.01 -0.06 -0.05 0.58  
## BLodds -0.25 -0.17 -0.06 0.32 0.24 0.05 -0.06 0.03 -0.08 0.06 -0.11  
## Podds -0.10 -0.10 -0.16 0.13 0.19 0.12 0.52 -0.11 -0.10 -0.06 0.11  
## BIBR Modds Godds BUodds Eodds BLodds Podds  
## bluff -0.09 -0.19 -0.18 -0.20 -0.17 -0.25 -0.10  
## BBwin -0.01 -0.13 -0.15 -0.12 -0.03 -0.17 -0.10  
## preFolds -0.14 -0.17 -0.13 -0.11 -0.18 -0.06 -0.16  
## Pot 0.26 0.05 0.17 0.27 0.00 0.32 0.13  
## Bets 0.32 0.15 0.27 0.22 0.18 0.24 0.19  
## Flop 0.19 0.11 0.16 0.11 0.16 0.05 0.12  
## PBR -0.10 -0.11 -0.07 -0.06 -0.09 -0.06 0.52  
## GBR -0.08 -0.02 0.58 0.01 0.01 0.03 -0.11  
## MrWBR -0.10 0.53 0.08 -0.04 -0.06 -0.08 -0.10  
## BuBR -0.04 -0.04 -0.04 0.62 -0.05 0.06 -0.06  
## EBR -0.17 0.05 -0.07 -0.09 0.58 -0.11 0.11  
## BIBR 1.00 -0.07 -0.08 -0.02 -0.06 0.54 -0.02  
## Modds -0.07 1.00 -0.05 -0.02 0.01 -0.01 -0.02  
## Godds -0.08 -0.05 1.00 0.03 0.00 0.01 -0.14  
## BUodds -0.02 -0.02 0.03 1.00 -0.04 0.06 -0.01  
## Eodds -0.06 0.01 0.00 -0.04 1.00 -0.02 0.03  
## BLodds 0.54 -0.01 0.01 0.06 -0.02 1.00 -0.02  
## Podds -0.02 -0.02 -0.14 -0.01 0.03 -0.02 1.00

Looking at the matrix above, Bluff has a positive correlation to BBwin, preFolds and PBR indicating these actions lead toward a bluff winning the game. The rest have negative correlations. Bill betting or raising has a most negative correlation to a bluff win, he may bluff less but he also had the highest correlation to the Pot size meaning he is a big better.

# Association Rules

Association rules can provide a set of conditions that led to a successful bluff. First to filter rules as below to Bluff=1. Also by setting Bets=1 it can filter out rules without bets that are less meaningful.

#data set for building association rules  
df3 <- data.frame(bluff=as.factor(df2$`Bluff?`),  
 pWin=as.factor(df2$`Pluribus Win?`),  
 Win=as.factor(df2$Winner),  
 Bets=as.factor(df2$AnyBets),  
 Pot=df2$Pot,   
 PBR=as.factor(df2$PBetsRaises),   
 OBR=as.factor(df2$OBetsRaises),   
 GBR=as.factor(df2$GBetsRaises),  
 MBR=as.factor(df2$MrWBetsRaises),  
 BuBR=as.factor(df2$BuBetsRaises),  
 EBR=as.factor(df2$EBetsRaises),  
 BiBR=as.factor(df2$BIBetsRaises),  
 Flop=as.factor(df2$`Has Flop`),  
 Modds=df2$Modds,  
 Godds=df2$Godds,  
 BUodds=df2$BUodds,  
 Eodds=df2$Eodds,  
 BLodds=df2$BLodds,  
 Podds=df2$Podds)  
#Apriori command builds rules with bluff hand wining the game  
myRules1 = apriori(data=df3, appearance = list(default="lhs", rhs="bluff=1"), parameter = list(supp = 0.001, conf = 0.6, maxlen = 4), control = list(verbose=F))  
#filter for rules with bets  
myRules1 <- subset(myRules1, lhs %in% c("Bets=1"))  
myRules1

## set of 83 rules

myRules1 <- sort(myRules1, decreasing=TRUE,by='support')  
#Rules with high support:  
inspect(myRules1[1:10])

## lhs rhs support   
## [1] {Win=Gogo, Bets=1, Godds=[0.16,0.21)} => {bluff=1} 0.04684318  
## [2] {Win=Bill, Bets=1, BLodds=[0.16,0.22)} => {bluff=1} 0.03869654  
## [3] {Win=Pluribus, Bets=1, Podds=[0.16,0.21)} => {bluff=1} 0.03054990  
## [4] {pWin=1, Bets=1, Podds=[0.16,0.21)} => {bluff=1} 0.03054990  
## [5] {Bets=1, GBR=1, Godds=[0.16,0.21)} => {bluff=1} 0.02647658  
## [6] {Bets=1, Pot=[100,250), OBR=1} => {bluff=1} 0.02647658  
## [7] {Bets=1, Pot=[100,250), PBR=0} => {bluff=1} 0.02647658  
## [8] {pWin=0, Bets=1, Pot=[100,250)} => {bluff=1} 0.02647658  
## [9] {Bets=1, BiBR=1, BLodds=[0.16,0.22)} => {bluff=1} 0.02647658  
## [10] {Bets=1, Pot=[100,250), EBR=0} => {bluff=1} 0.02443992  
## confidence coverage lift count  
## [1] 1.0000000 0.04684318 3.748092 23   
## [2] 0.9047619 0.04276986 3.391130 19   
## [3] 1.0000000 0.03054990 3.748092 15   
## [4] 1.0000000 0.03054990 3.748092 15   
## [5] 0.6842105 0.03869654 2.564484 13   
## [6] 0.6500000 0.04073320 2.436260 13   
## [7] 0.6500000 0.04073320 2.436260 13   
## [8] 0.6500000 0.04073320 2.436260 13   
## [9] 0.8125000 0.03258656 3.045324 13   
## [10] 0.6000000 0.04073320 2.248855 12

This returns 83 rules of which are dominated with rules having odds. It solidifies Gogo, then Bill, then Pluribus as the Buffers with highest support scores. Next, Lets filter out the odds data to see more interesting Bluffing associations.

#data set for building association rules  
df3 <- data.frame(bluff=as.factor(df2$`Bluff?`),  
 pWin=as.factor(df2$`Pluribus Win?`),  
 Win=as.factor(df2$Winner),  
 Bets=as.factor(df2$AnyBets),  
 Pot=df2$Pot,   
 PBR=as.factor(df2$PBetsRaises),   
 OBR=as.factor(df2$OBetsRaises),   
 GBR=as.factor(df2$GBetsRaises),  
 MBR=as.factor(df2$MrWBetsRaises),  
 BuBR=as.factor(df2$BuBetsRaises),  
 EBR=as.factor(df2$EBetsRaises),  
 BiBR=as.factor(df2$BIBetsRaises),  
 Flop=as.factor(df2$`Has Flop`)  
 #Modds=df$Modds,  
 #Godds=df$Godds,  
 #BUodds=df$BUodds,  
 #Eodds=df$Eodds,  
 #BLodds=df$BLodds,  
 #Podds=df$Podds)  
)  
#Apriori command builds rules with bluff hand wining the game  
myRules1 = apriori(data=df3, appearance = list(default="lhs", rhs="bluff=1"), parameter = list(supp = 0.001, conf = 0.6, maxlen = 4), control = list(verbose=F))  
myRules1

## set of 711 rules

plot(myRules1, engine = "htmlwidget")

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.

myRules1 <- subset(myRules1, lhs %in% c("Bets=1"))  
myRules1

## set of 39 rules

myRules1 <- sort(myRules1, decreasing=TRUE,by='support')  
#Rules with high support:  
inspect(myRules1[1:10])

## lhs rhs support confidence  
## [1] {Bets=1, Pot=[100,250), OBR=1} => {bluff=1} 0.02647658 0.6500000   
## [2] {Bets=1, Pot=[100,250), PBR=0} => {bluff=1} 0.02647658 0.6500000   
## [3] {pWin=0, Bets=1, Pot=[100,250)} => {bluff=1} 0.02647658 0.6500000   
## [4] {Bets=1, Pot=[100,250), EBR=0} => {bluff=1} 0.02443992 0.6000000   
## [5] {pWin=0, Bets=1, PBR=2} => {bluff=1} 0.01221996 0.6666667   
## [6] {Bets=1, Pot=[100,250), GBR=1} => {bluff=1} 0.01018330 0.7142857   
## [7] {Bets=1, Pot=[100,250), BiBR=1} => {bluff=1} 0.01018330 0.6250000   
## [8] {Win=Bill, Bets=1, Pot=[100,250)} => {bluff=1} 0.01018330 0.6250000   
## [9] {Win=Gogo, Bets=1, Pot=[100,250)} => {bluff=1} 0.01018330 0.7142857   
## [10] {Bets=1, OBR=4, EBR=2} => {bluff=1} 0.00814664 0.6666667   
## coverage lift count  
## [1] 0.04073320 2.436260 13   
## [2] 0.04073320 2.436260 13   
## [3] 0.04073320 2.436260 13   
## [4] 0.04073320 2.248855 12   
## [5] 0.01832994 2.498728 6   
## [6] 0.01425662 2.677208 5   
## [7] 0.01629328 2.342557 5   
## [8] 0.01629328 2.342557 5   
## [9] 0.01425662 2.677208 5   
## [10] 0.01221996 2.498728 4

myRules1 <- sort(myRules1, decreasing=TRUE,by='confidence')  
#Rules with high confidence:  
inspect(myRules1[1:10])

## lhs rhs support confidence  
## [1] {Bets=1, OBR=3, MBR=3} => {bluff=1} 0.00407332 1   
## [2] {pWin=0, Bets=1, PBR=3} => {bluff=1} 0.00407332 1   
## [3] {Bets=1, PBR=2, GBR=2} => {bluff=1} 0.00407332 1   
## [4] {Bets=1, BuBR=1, BiBR=2} => {bluff=1} 0.00407332 1   
## [5] {Bets=1, Pot=[100,250), Flop=1} => {bluff=1} 0.00407332 1   
## [6] {Bets=1, PBR=3, EBR=3} => {bluff=1} 0.00203666 1   
## [7] {Bets=1, PBR=1, EBR=3} => {bluff=1} 0.00203666 1   
## [8] {Bets=1, PBR=3, OBR=3} => {bluff=1} 0.00203666 1   
## [9] {Win=Eddie, Bets=1, PBR=3} => {bluff=1} 0.00203666 1   
## [10] {Win=Gogo, Bets=1, PBR=3} => {bluff=1} 0.00203666 1   
## coverage lift count  
## [1] 0.00407332 3.748092 2   
## [2] 0.00407332 3.748092 2   
## [3] 0.00407332 3.748092 2   
## [4] 0.00407332 3.748092 2   
## [5] 0.00407332 3.748092 2   
## [6] 0.00203666 3.748092 1   
## [7] 0.00203666 3.748092 1   
## [8] 0.00203666 3.748092 1   
## [9] 0.00203666 3.748092 1   
## [10] 0.00203666 3.748092 1

myRules1 <- sort(myRules1, decreasing=TRUE,by='lift')  
#Rules with high lift:  
inspect(myRules1[1:10])

## lhs rhs support confidence  
## [1] {Bets=1, OBR=3, MBR=3} => {bluff=1} 0.00407332 1   
## [2] {pWin=0, Bets=1, PBR=3} => {bluff=1} 0.00407332 1   
## [3] {Bets=1, PBR=2, GBR=2} => {bluff=1} 0.00407332 1   
## [4] {Bets=1, BuBR=1, BiBR=2} => {bluff=1} 0.00407332 1   
## [5] {Bets=1, Pot=[100,250), Flop=1} => {bluff=1} 0.00407332 1   
## [6] {Bets=1, PBR=3, EBR=3} => {bluff=1} 0.00203666 1   
## [7] {Bets=1, PBR=1, EBR=3} => {bluff=1} 0.00203666 1   
## [8] {Bets=1, PBR=3, OBR=3} => {bluff=1} 0.00203666 1   
## [9] {Win=Eddie, Bets=1, PBR=3} => {bluff=1} 0.00203666 1   
## [10] {Win=Gogo, Bets=1, PBR=3} => {bluff=1} 0.00203666 1   
## coverage lift count  
## [1] 0.00407332 3.748092 2   
## [2] 0.00407332 3.748092 2   
## [3] 0.00407332 3.748092 2   
## [4] 0.00407332 3.748092 2   
## [5] 0.00407332 3.748092 2   
## [6] 0.00203666 3.748092 1   
## [7] 0.00203666 3.748092 1   
## [8] 0.00203666 3.748092 1   
## [9] 0.00203666 3.748092 1   
## [10] 0.00203666 3.748092 1

plot(myRules1, engine = "htmlwidget")

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.

Above show two rule set plots first with 711 rules and then down to the 39 rules after filtering for games with bets. The rules with the most lift also have high confidence (top area are redder). Lift indicates how predictable the confidence is, meaning they are more reliable rules. These are good indicators of bluffing, players can identify and utilize to either counter bet or bluff themselves. If you zoom into the high confidence rules you’ll see some interesting associations. When pWin=0,and PBR=3 yet the game was a bluff win meaning players are successfully bluffing against Pluribus here. Other rules that raise up include people betting against Mr. White (MBR=3 OBR=3) and Budd betting against Bill (BuBR=1 BiBR=2).

# Modeling

Here the exercise uses SVM and Random Forest modeling techniques to predict and analyze bluffing patterns. First SVM modelling.

## SVM

The first model will predict if the game will be won with a bluff hand using all data available.

bluffSet$bluff <- as.factor(bluffSet$bluff)  
#set training and test data sets  
set.seed(10)  
trainList <- createDataPartition(y=bluffSet$bluff,p=.70, list=FALSE) #344-147 int 1:344  
trainSet <- bluffSet[trainList,]  
testSet <- bluffSet[-trainList,]  
#k folding  
# value of K equal to 10  
train\_control <- trainControl(method = "cv",number = 10)  
#build svm radial model to predict if the game has a bluff. uses the carets train command with available models here :https://rdrr.io/cran/caret/man/models.html  
model.ksvm <- train(bluff~., data=trainSet, method="svmRadial",   
 preProc=c("center","scale"),   
 trControl = train\_control  
 )  
svmPred <- predict(model.ksvm, newdata=testSet, type = "raw")  
cm<-confusionMatrix(svmPred,reference = testSet$bluff)  
cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 105 19  
## 1 3 20  
##   
## Accuracy : 0.8503   
## 95% CI : (0.7822, 0.9038)  
## No Information Rate : 0.7347   
## P-Value [Acc > NIR] : 0.0005857   
##   
## Kappa : 0.5582   
##   
## Mcnemar's Test P-Value : 0.0013838   
##   
## Sensitivity : 0.9722   
## Specificity : 0.5128   
## Pos Pred Value : 0.8468   
## Neg Pred Value : 0.8696   
## Prevalence : 0.7347   
## Detection Rate : 0.7143   
## Detection Prevalence : 0.8435   
## Balanced Accuracy : 0.7425   
##   
## 'Positive' Class : 0   
##

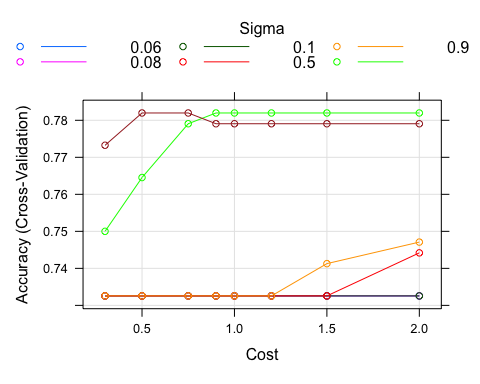
The accuracy of the first model comes out to 85% (less with tuning) which means when using all 17 fields in the bluffset the model can predict that the poker game would be won with a Bluff (Winning hand has <= 20% odds of winning). #85% accuracy sounds quite accurate but a player in a game won’t know everything in our dataset especially odds of others player hands.

Next the exercise will build a model with what a player might know from watching the table, and after rerunning models this model below performed the best. This model specifically looks at a players hand and how many bets are being made. Adding other vectors actually reduced accuracy. Below we see a model based on Gogo’s hand and how many bets they see.

train\_control <- trainControl(method = "cv",number = 2)  
model.ksvm2 <- train(bluff~Godds+Bets, data=trainSet, method="svmRadial",   
 preProc=c("center","scale"),   
 trControl = train\_control,  
 tuneGrid = expand.grid(sigma = c(0.02, 0.04, 0.06, 0.08, 0.1, 0.5, 0.9),  
 C = c(0.3, 0.5, 0.75, 0.9, 1, 1.2, 1.5, 2 )))  
svmPred2 <- predict(model.ksvm2, newdata=testSet, type = "raw")  
#display accuracy of predictions svmRadial .795 svmPoly .789, svmLinear, svmLinearWeights ~.734  
cm2<-confusionMatrix(svmPred2,reference = testSet$bluff)  
cm2

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 105 27  
## 1 3 12  
##   
## Accuracy : 0.7959   
## 95% CI : (0.7217, 0.8579)  
## No Information Rate : 0.7347   
## P-Value [Acc > NIR] : 0.05329   
##   
## Kappa : 0.3484   
##   
## Mcnemar's Test P-Value : 2.679e-05   
##   
## Sensitivity : 0.9722   
## Specificity : 0.3077   
## Pos Pred Value : 0.7955   
## Neg Pred Value : 0.8000   
## Prevalence : 0.7347   
## Detection Rate : 0.7143   
## Detection Prevalence : 0.8980   
## Balanced Accuracy : 0.6400   
##   
## 'Positive' Class : 0   
##

plot(model.ksvm2)

 The plot above shows how tuning can effect accuracy. Default tuning does a fine job in this case. 79.5% accuracy is enough that a player could make more informed decisions.

Next to build a model with what minimal information a player might know preflop, no knowledge of bets/raises, and after rerunning models this model performed the best. It predicts bluff using Gogo’s hand, whether there is a flop and if people are folding. (no betting data)

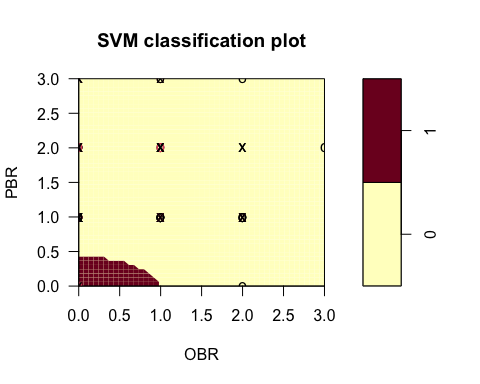
model.ksvm3 <- train(bluff~Godds+Flop+preFolds, data=trainSet, method="svmRadial"  
 , preProc=c("center","scale")  
 , trControl = train\_control)  
svmPred3 <- predict(model.ksvm3, newdata=testSet, type = "raw")  
#display accuracy of predictions svmRadial .734, which is actually midly affected by tuning:  
cm3<-confusionMatrix(svmPred3,reference = testSet$bluff)  
cm3

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 108 39  
## 1 0 0  
##   
## Accuracy : 0.7347   
## 95% CI : (0.6556, 0.804)  
## No Information Rate : 0.7347   
## P-Value [Acc > NIR] : 0.543   
##   
## Kappa : 0   
##   
## Mcnemar's Test P-Value : 1.166e-09   
##   
## Sensitivity : 1.0000   
## Specificity : 0.0000   
## Pos Pred Value : 0.7347   
## Neg Pred Value : NaN   
## Prevalence : 0.7347   
## Detection Rate : 0.7347   
## Detection Prevalence : 1.0000   
## Balanced Accuracy : 0.5000   
##   
## 'Positive' Class : 0   
##

With such little information the model above hits a low accuracy of 73%.

Other analysis SVM models can do with this data is compare how a given player is doing during a bluff win vs a bluff loss (someone else won with a bluff). Below looks at scenarios Pluribus won with a bluff and scenarios Pluribus lost to a bluff.

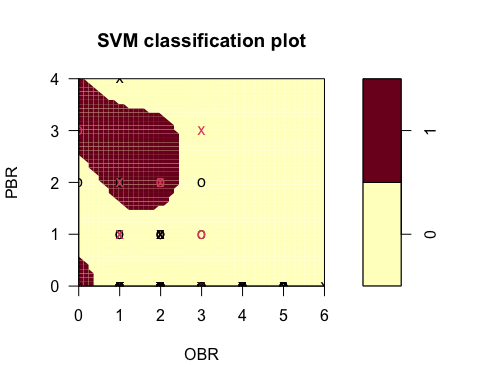
dfPloss <- filter(df2, `Pluribus Win?` == 1)  
df4 <- data.frame(bluff=as.factor(dfPloss$`Bluff?`),PBR=as.numeric(dfPloss$PBetsRaises),OBR=as.numeric(dfPloss$OBetsRaises))  
set.seed(10)  
trainList2 <- createDataPartition(y=df4$bluff,p=.90, list=FALSE) #344-147 int 1:344  
trainSet2 <- df4[trainList2,]  
testSet2 <- df4[-trainList2,]  
  
#df5 <- data.frame(bluff=testSet$bluff,PBR=as.numeric(testSet$PBR),OBR=as.numeric(testSet2$OBR))  
svm\_model<- svm(bluff ~ PBR+OBR,   
 data = trainSet2,   
 type = "C-classification",   
 kernel = "radial",  
 cost = 5,  
 scale = FALSE)  
plot(svm\_model, df4)



svmPred <- predict(svm\_model, newdata=testSet2, type = "raw")  
#display accuracy of predictions  
confusionMatrix(svmPred, reference = testSet2$bluff)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 5 1  
## 1 1 1  
##   
## Accuracy : 0.75   
## 95% CI : (0.3491, 0.9681)  
## No Information Rate : 0.75   
## P-Value [Acc > NIR] : 0.6785   
##   
## Kappa : 0.3333   
##   
## Mcnemar's Test P-Value : 1.0000   
##   
## Sensitivity : 0.8333   
## Specificity : 0.5000   
## Pos Pred Value : 0.8333   
## Neg Pred Value : 0.5000   
## Prevalence : 0.7500   
## Detection Rate : 0.6250   
## Detection Prevalence : 0.7500   
## Balanced Accuracy : 0.6667   
##   
## 'Positive' Class : 0   
##

dfPloss <- filter(df2, `Pluribus Win?` == 0)  
df4 <- data.frame(bluff=as.factor(dfPloss$`Bluff?`),PBR=as.numeric(dfPloss$PBetsRaises),OBR=as.numeric(dfPloss$OBetsRaises))  
set.seed(10)  
trainList2 <- createDataPartition(y=df4$bluff,p=.90, list=FALSE) #344-147 int 1:344  
trainSet2 <- df4[trainList2,]  
testSet2 <- df4[-trainList2,]  
  
#df5 <- data.frame(bluff=testSet$bluff,PBR=as.numeric(testSet$PBR),OBR=as.numeric(testSet2$OBR))  
svm\_model<- svm(bluff ~ PBR+OBR,   
 data = trainSet2,   
 type = "C-classification",   
 kernel = "radial",  
 cost = 5,  
 scale = FALSE)  
plot(svm\_model, df4)



svmPred <- predict(svm\_model, newdata=testSet2, type = "raw")  
#display accuracy of predictions  
confusionMatrix(svmPred, reference = testSet2$bluff)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 28 8  
## 1 1 3  
##   
## Accuracy : 0.775   
## 95% CI : (0.6155, 0.8916)  
## No Information Rate : 0.725   
## P-Value [Acc > NIR] : 0.3048   
##   
## Kappa : 0.2969   
##   
## Mcnemar's Test P-Value : 0.0455   
##   
## Sensitivity : 0.9655   
## Specificity : 0.2727   
## Pos Pred Value : 0.7778   
## Neg Pred Value : 0.7500   
## Prevalence : 0.7250   
## Detection Rate : 0.7000   
## Detection Prevalence : 0.9000   
## Balanced Accuracy : 0.6191   
##   
## 'Positive' Class : 0   
##

The plots above illustrate areas of Pluribus betting (PBR) against other betting (OBR). First plot shows bluff wins by Pluribus. Here, at the lower left red edge betting between 0-1 is where bluffing wins occurred. This means there were likely very few bluffing by Pluribus when there were lots of bets by anyone. The second plot shows when Pluribus lost to bluffing. This plot shows a new red area where others were winning with bluffs against Pluribus bets. Is Pluribus being taken advantage of here? These models had 75% and 77% accuracy respectively.

## Random Forest

### Predicting Pluribus’ Bluff

#select columns for model and create a factor  
pluribusBluffDf <- data.frame(PBluff=df$PBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 OBR = df$OBetsRaises)  
  
trainList <- createDataPartition(y=pluribusBluffDf$PBluff,p=.70, list=FALSE)  
trainSet <- pluribusBluffDf[trainList,]  
testSet <- pluribusBluffDf[-trainList,]  
prfm <- randomForest(PBluff~., data=trainSet, ntree=10)  
print(prfm)

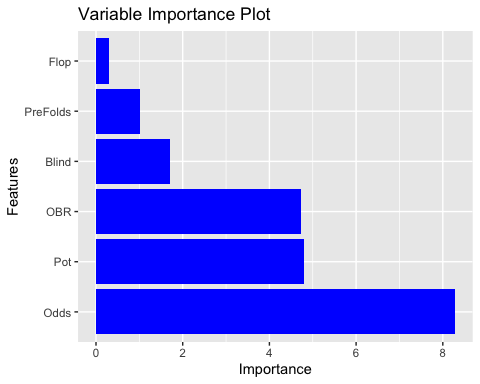
##   
## Call:  
## randomForest(formula = PBluff ~ ., data = trainSet, ntree = 10)   
## Type of random forest: classification  
## Number of trees: 10  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 4.06%  
## Confusion matrix:  
## 0 1 class.error  
## 0 329 3 0.009036145  
## 1 11 2 0.846153846

pRF <- predict(prfm, testSet, type=c("class"))  
confusionMatrix(pRF,reference = testSet$PBluff)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 140 4  
## 1 1 1  
##   
## Accuracy : 0.9658   
## 95% CI : (0.9219, 0.9888)  
## No Information Rate : 0.9658   
## P-Value [Acc > NIR] : 0.6160   
##   
## Kappa : 0.2715   
##   
## Mcnemar's Test P-Value : 0.3711   
##   
## Sensitivity : 0.9929   
## Specificity : 0.2000   
## Pos Pred Value : 0.9722   
## Neg Pred Value : 0.5000   
## Prevalence : 0.9658   
## Detection Rate : 0.9589   
## Detection Prevalence : 0.9863   
## Balanced Accuracy : 0.5965   
##   
## 'Positive' Class : 0   
##

While this model performed with high accuracy, it was very bad at actually predicting the bluff. Much of this error is likely due to the nature of the sample data - Pluribus (and players in this data set in general) did not bluff particularly often. There is possibility for better results to be obtained through tuning.

var\_imp <- varImp(prfm)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")



Given the massive importance the model placed on Odds seems likely to be “cheating” as any odds over 20% are automatically not considered a bluff. This will be filtered to more appropriately factor for those odds. In a real environment, another player would not know your odds, and this would not be a practical indicator of how to know if someone is bluffing. On the other hand, it could provide some insight as to what kinds of hands may be the best to bluff with. Beyond this, the model itself will be tuned with slightly different parameters, including weighting the bluffed hands to appear more often in a cross validation approach to model training to attempt to build a more robust model for detecting bluffs.

pluribusBluffDf <- data.frame(PBluff=df$PBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 OBR = df$OBetsRaises)  
  
pluribusBluffDf <- pluribusBluffDf %>% filter(Odds <= .2)  
  
pluribusBluffDf <- pluribusBluffDf %>% mutate\_if(is.factor,as.integer)  
glimpse(pluribusBluffDf)

## Rows: 311  
## Columns: 7  
## $ PBluff <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1…  
## $ Pot <dbl> 200, 450, 4250, 600, 250, 2500, 1550, 100, 250, 750, 100, 250…  
## $ Flop <int> 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 2…  
## $ Blind <int> 2, 3, 1, 6, 4, 2, 3, 2, 1, 5, 4, 2, 6, 4, 2, 3, 6, 5, 4, 2, 3…  
## $ Odds <dbl> 0.12, 0.13, 0.20, 0.15, 0.20, 0.18, 0.19, 0.15, 0.14, 0.17, 0…  
## $ PreFolds <int> 5, 4, 4, 5, 5, 4, 4, 5, 5, 4, 5, 5, 4, 5, 5, 5, 5, 4, 5, 4, 4…  
## $ OBR <int> 2, 3, 5, 3, 2, 4, 4, 1, 2, 4, 1, 2, 5, 1, 2, 1, 2, 3, 2, 5, 4…

x <- pluribusBluffDf[, -1]  
pluribusBluffDf$PBluff <- factor(pluribusBluffDf$PBluff)  
levels(pluribusBluffDf$PBluff) <- make.names(levels(pluribusBluffDf$PBluff))  
y <- pluribusBluffDf$PBluff  
  
train\_control <- trainControl(  
 method = 'cv',  
 number = 10,  
 classProbs = TRUE,   
 sampling = 'down',  
 )  
  
# Define the class weights  
classWeights <- c(1, 5)  
  
pluribusRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
pluribusRFTune

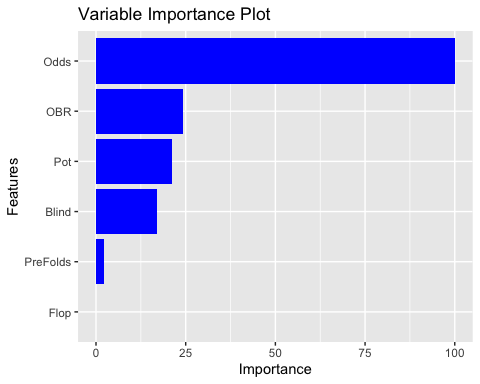
## Random Forest   
##   
## 311 samples  
## 7 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 280, 279, 280, 280, 279, 280, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.8424126 0.3458765  
## 4 0.8204301 0.3298372  
## 6 0.8300134 0.2833579  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 2.

pRF <- predict(pluribusRFTune, pluribusBluffDf, type=c("raw"))  
confusionMatrix(pRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 227 0  
## X2 66 18  
##   
## Accuracy : 0.7878   
## 95% CI : (0.7381, 0.8319)  
## No Information Rate : 0.9421   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.2848   
##   
## Mcnemar's Test P-Value : 1.235e-15   
##   
## Sensitivity : 0.7747   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.2143   
## Prevalence : 0.9421   
## Detection Rate : 0.7299   
## Detection Prevalence : 0.7299   
## Balanced Accuracy : 0.8874   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(pluribusRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.

 Unfortunately, this model performed worse than the first. The importance of PreFolds and whether or not Pluribus had the flop dropped out entirely. These columns will be dropped in an attempt to thwart any negative affects the added dimensionality of these columns had on the model. It is encouraging to see that odds still play a role after adjusting accordingly. It should be noted that there is a possibility that the first model only performed better because the test data provided such a small sample of bluffed hands to predict. It is likely that this model is more true to reality.

pluribusBluffDf <- data.frame(PBluff=df$PBluff,   
 Pot=df$Pot,   
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 OBR = df$OBetsRaises)  
  
pluribusBluffDf <- pluribusBluffDf %>% filter(Odds <= .2)  
  
pluribusBluffDf <- pluribusBluffDf %>% mutate\_if(is.factor,as.integer)

x <- pluribusBluffDf[, -1]  
pluribusBluffDf$PBluff <- factor(pluribusBluffDf$PBluff)  
levels(pluribusBluffDf$PBluff) <- make.names(levels(pluribusBluffDf$PBluff))  
y <- pluribusBluffDf$PBluff  
  
pluribusRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
pluribusRFTune

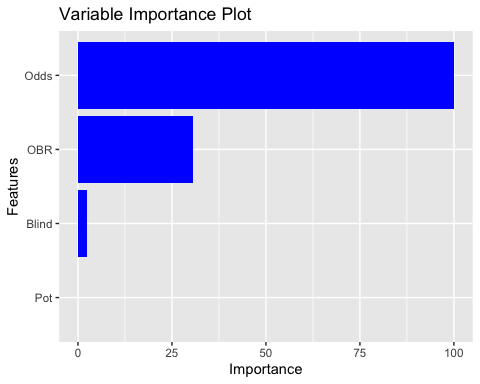
## Random Forest   
##   
## 311 samples  
## 5 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 280, 280, 280, 280, 280, 280, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.7939315 0.2678648  
## 3 0.8291868 0.3353324  
## 4 0.7871371 0.2481137  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 3.

pRF <- predict(pluribusRFTune, pluribusBluffDf, type=c("raw"))  
confusionMatrix(pRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 242 0  
## X2 51 18  
##   
## Accuracy : 0.836   
## 95% CI : (0.7901, 0.8754)  
## No Information Rate : 0.9421   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3545   
##   
## Mcnemar's Test P-Value : 2.534e-12   
##   
## Sensitivity : 0.8259   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.2609   
## Prevalence : 0.9421   
## Detection Rate : 0.7781   
## Detection Prevalence : 0.7781   
## Balanced Accuracy : 0.9130   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(pluribusRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



Further, accuracy was lost and this model was even worse at predicting the bluff. The blind column additionally lost all importance. It will now be dropped.

pluribusBluffDf <- data.frame(PBluff=df$PBluff,   
 Pot=df$Pot,   
 Odds = df$POdds,  
 OBR = df$OBetsRaises)  
  
pluribusBluffDf <- pluribusBluffDf %>% filter(Odds <= .2)  
  
pluribusBluffDf <- pluribusBluffDf %>% mutate\_if(is.factor,as.integer)

x <- pluribusBluffDf[, -1]  
pluribusBluffDf$PBluff <- factor(pluribusBluffDf$PBluff)  
levels(pluribusBluffDf$PBluff) <- make.names(levels(pluribusBluffDf$PBluff))  
y <- pluribusBluffDf$PBluff  
  
pluribusRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )

## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .

pluribusRFTune

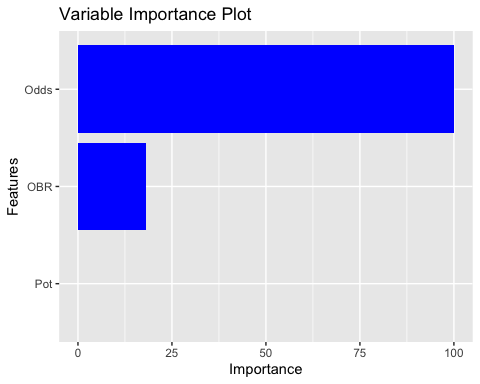
## Random Forest   
##   
## 311 samples  
## 4 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 279, 280, 280, 280, 280, 280, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.8073387 0.2803846  
## 3 0.7807997 0.2595736  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 2.

pRF <- predict(pluribusRFTune, pluribusBluffDf, type=c("raw"))  
confusionMatrix(pRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 230 0  
## X2 63 18  
##   
## Accuracy : 0.7974   
## 95% CI : (0.7484, 0.8407)  
## No Information Rate : 0.9421   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.2971   
##   
## Mcnemar's Test P-Value : 5.662e-15   
##   
## Sensitivity : 0.7850   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.2222   
## Prevalence : 0.9421   
## Detection Rate : 0.7395   
## Detection Prevalence : 0.7395   
## Balanced Accuracy : 0.8925   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(pluribusRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



This model had identical results. But the pot’s importance dropped out entirely. This seems to be counter-intuitive to a typical player’s mentality. It seems the best model of the random forests would be the first option. This technique will be employed across all players.

### Predicting Gogo’s Bluff

gogoBluffDf <- data.frame(GBluff=df$GBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 PBR = df$PBetsRaises,   
 MBR = df$MrWBetsRaises,   
 BuBR = df$BuBetsRaises,   
 EBR = df$EBetsRaises,   
 BiBR = df$BiBetsRaises)  
  
gogoBluffDf <- gogoBluffDf %>% filter(Odds <= .2)  
  
gogoBluffDf <- gogoBluffDf %>% mutate\_if(is.factor,as.integer)

x <- gogoBluffDf[, -1]  
gogoBluffDf$GBluff <- factor(gogoBluffDf$GBluff)  
levels(gogoBluffDf$GBluff) <- make.names(levels(gogoBluffDf$GBluff))  
y <- gogoBluffDf$GBluff  
  
gogoRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
gogoRFTune

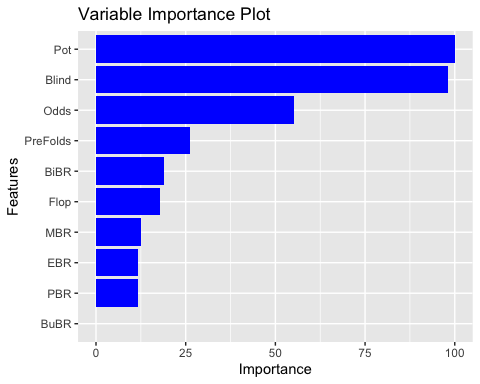
## Random Forest   
##   
## 311 samples  
## 11 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 279, 280, 280, 280, 280, 280, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.6175403 0.11756076  
## 6 0.5790323 0.10638477  
## 10 0.5914315 0.04920434  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 2.

gRF <- predict(gogoRFTune, gogoBluffDf, type=c("raw"))  
confusionMatrix(gRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 181 0  
## X2 108 22  
##   
## Accuracy : 0.6527   
## 95% CI : (0.597, 0.7056)  
## No Information Rate : 0.9293   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.1917   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.6263   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.1692   
## Prevalence : 0.9293   
## Detection Rate : 0.5820   
## Detection Prevalence : 0.5820   
## Balanced Accuracy : 0.8131   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(gogoRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



### Predicting MrWhite’s Bluffs

mrwBluffDf <- data.frame(MBluff=df$MrWBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 PBR = df$PBetsRaises,   
 GBR = df$GBetsRaises,   
 BuBR = df$BuBetsRaises,   
 EBR = df$EBetsRaises,   
 BiBR = df$BiBetsRaises)  
  
mrwBluffDf <- mrwBluffDf %>% filter(Odds <= .2)  
  
mrwBluffDf <- mrwBluffDf %>% mutate\_if(is.factor,as.integer)

x <- mrwBluffDf[, -1]  
mrwBluffDf$MBluff <- factor(mrwBluffDf$MBluff)  
levels(mrwBluffDf$MBluff) <- make.names(levels(mrwBluffDf$MBluff))  
y <- mrwBluffDf$MBluff  
  
mrwRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
mrwRFTune

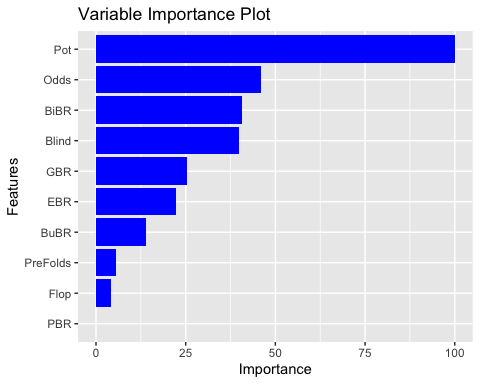
## Random Forest   
##   
## 311 samples  
## 11 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 279, 279, 281, 280, 280, 280, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.5277285 0.01518093  
## 6 0.5631384 0.04769942  
## 10 0.5267137 0.04948895  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 6.

mRF <- predict(mrwRFTune, mrwBluffDf, type=c("raw"))  
confusionMatrix(mRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 153 0  
## X2 142 16  
##   
## Accuracy : 0.5434   
## 95% CI : (0.4863, 0.5997)  
## No Information Rate : 0.9486   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0998   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.5186   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.1013   
## Prevalence : 0.9486   
## Detection Rate : 0.4920   
## Detection Prevalence : 0.4920   
## Balanced Accuracy : 0.7593   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(mrwRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



### Predicting Budd’s Bluff

buBluffDf <- data.frame(BBluff=df$BuBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 PBR = df$PBetsRaises,   
 MBR = df$MrWBetsRaises,   
 GBR = df$GBetsRaises,   
 EBR = df$EBetsRaises,   
 BiBR = df$BiBetsRaises)  
  
buBluffDf <- buBluffDf %>% filter(Odds <= .2)  
  
buBluffDf <- buBluffDf %>% mutate\_if(is.factor,as.integer)

x <- buBluffDf[, -1]  
buBluffDf$BBluff <- factor(buBluffDf$BBluff)  
levels(buBluffDf$BBluff) <- make.names(levels(buBluffDf$BBluff))  
y <- buBluffDf$BBluff  
  
buRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
buRFTune

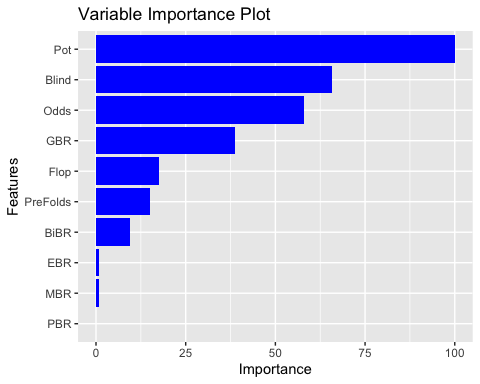
## Random Forest   
##   
## 311 samples  
## 11 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 280, 280, 280, 280, 279, 281, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.5056586 0.008385826  
## 6 0.5300134 0.022110847  
## 10 0.4890054 0.019999835  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 6.

bRF <- predict(buRFTune, buBluffDf, type=c("raw"))  
confusionMatrix(bRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 175 0  
## X2 124 12  
##   
## Accuracy : 0.6013   
## 95% CI : (0.5445, 0.6561)  
## No Information Rate : 0.9614   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0982   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.58528   
## Specificity : 1.00000   
## Pos Pred Value : 1.00000   
## Neg Pred Value : 0.08824   
## Prevalence : 0.96141   
## Detection Rate : 0.56270   
## Detection Prevalence : 0.56270   
## Balanced Accuracy : 0.79264   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(buRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



### Predicting Eddie’s Bluff

edBluffDf <- data.frame(EBluff=df$EBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 PBR = df$PBetsRaises,   
 MBR = df$MrWBetsRaises,   
 BuBR = df$BuBetsRaises,   
 GBR = df$GBetsRaises,   
 BiBR = df$BiBetsRaises)  
  
edBluffDf <- edBluffDf %>% filter(Odds <= .2)  
  
edBluffDf <- edBluffDf %>% mutate\_if(is.factor,as.integer)

x <- edBluffDf[, -1]  
edBluffDf$EBluff <- factor(edBluffDf$EBluff)  
levels(edBluffDf$EBluff) <- make.names(levels(edBluffDf$EBluff))  
y <- edBluffDf$EBluff  
  
edRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
edRFTune

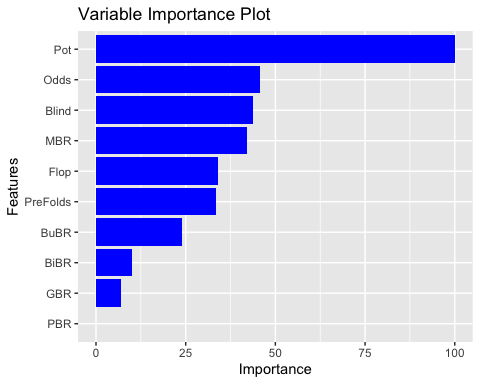
## Random Forest   
##   
## 311 samples  
## 11 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 280, 279, 280, 280, 280, 280, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.4307191 0.01717550  
## 6 0.5372917 0.04117295  
## 10 0.4946102 0.04378354  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 6.

eRF <- predict(edRFTune, edBluffDf, type=c("raw"))  
confusionMatrix(eRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 115 0  
## X2 187 9  
##   
## Accuracy : 0.3987   
## 95% CI : (0.3439, 0.4555)  
## No Information Rate : 0.9711   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0344   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.38079   
## Specificity : 1.00000   
## Pos Pred Value : 1.00000   
## Neg Pred Value : 0.04592   
## Prevalence : 0.97106   
## Detection Rate : 0.36977   
## Detection Prevalence : 0.36977   
## Balanced Accuracy : 0.69040   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(edRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



### Predicting Bill’s Bluff

biBluffDf <- data.frame(BBluff=df$BiBluff,   
 Pot=df$Pot,   
 Flop=df$HasFlop,  
 Blind=df$HasBlind,  
 Odds = df$POdds,  
 PreFolds = df$PreFolds,  
 PBR = df$PBetsRaises,   
 MBR = df$MrWBetsRaises,   
 GBR = df$GBetsRaises,   
 EBR = df$EBetsRaises,   
 BuBR = df$BuBetsRaises)  
  
biBluffDf <- biBluffDf %>% filter(Odds <= .2)  
  
biBluffDf <- biBluffDf %>% mutate\_if(is.factor,as.integer)

x <- biBluffDf[, -1]  
biBluffDf$BBluff <- factor(biBluffDf$BBluff)  
levels(biBluffDf$BBluff) <- make.names(levels(biBluffDf$BBluff))  
y <- biBluffDf$BBluff  
  
biRFTune <- train(  
 x = x,  
 y = y,  
 method = "rf",  
 metric = "Accuracy",  
 trControl = train\_control,  
 weights = ifelse(y == 'X1', classWeights[1], classWeights[2])  
 )  
biRFTune

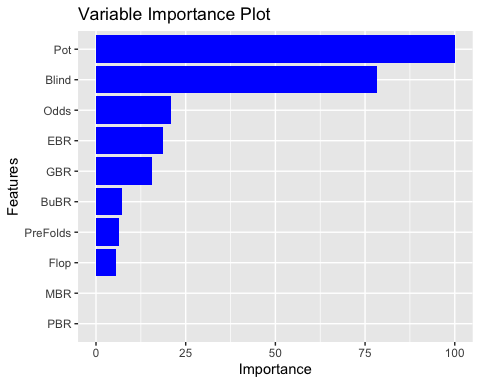
## Random Forest   
##   
## 311 samples  
## 11 predictor  
## 2 classes: 'X1', 'X2'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 281, 280, 279, 280, 281, 279, ...   
## Addtional sampling using down-sampling  
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.5846909 0.10356552  
## 6 0.5585618 0.07343711  
## 10 0.5996976 0.07835789  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 10.

bRF <- predict(biRFTune, biBluffDf, type=c("raw"))  
confusionMatrix(bRF,reference = y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction X1 X2  
## X1 155 0  
## X2 139 17  
##   
## Accuracy : 0.5531   
## 95% CI : (0.4959, 0.6092)  
## No Information Rate : 0.9453   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.1087   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.5272   
## Specificity : 1.0000   
## Pos Pred Value : 1.0000   
## Neg Pred Value : 0.1090   
## Prevalence : 0.9453   
## Detection Rate : 0.4984   
## Detection Prevalence : 0.4984   
## Balanced Accuracy : 0.7636   
##   
## 'Positive' Class : X1   
##

var\_imp <- varImp(biRFTune)  
  
# Plot the feature importance  
ggplot(var\_imp, aes(x = reorder(rownames(var\_imp), -Overall), y = Overall)) +   
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Variable Importance Plot") +  
 xlab("Features") +  
 ylab("Importance")

## Coordinate system already present. Adding new coordinate system, which will  
## replace the existing one.



# Results

For bluff predictions generally, model accuracy varied widely among these players, and it is usually better to just assume that they are not bluffing than to try to guess their bluff. In every instance, the “no information rate” of the model performs better than the model’s accuracy, usually by 20-30% but also potentially as high as 60% in the case of Eddie. Eddie’s unpredictability may potentially be a factor into why he is able to obtain so many winnings from his bluffs. Suggested follow up analysis would involve winning pot amounts for bluffs to identify whether better bluffing strategies focus on a few big pots or several small pots. The gap between model accuracy and the no information rate is driven by the truly abysmal recall scores in every model. The class imbalance and scarcity of bluffs across the data compared to non-bluffed hands leads the model to naturally predict a lot of non-bluffs, and even weighting bluffed hands in cross validation in an attempt to account for this was unable to satisfactorily resolve this discrepancy. This is unfortunate, but it highlights the unpredictability of people. Along these lines, it is interesting to see how some human players are slightly more predictable than others - Gogo has a slightly more identifiable pattern as her model has an accuracy score 5-10% better at predicting her bluff than other players. This is particularly noteworthy as, given the relative frequency with which she bluffs, this is an unexpected result. Similarly, it is interesting, though perhaps expected, to see that Pluribus is the most predictable player in the game with its model accuracy scoring well above even Gogo’s model.

More positively, the SVM models and association rule mining were able to identify some more positive strategic direction. More specifically, Pluribus is particularly weak to bluffing compared with other players and it seems that human capacity for bluffing may still give a competitive advantage against machine players.

While it is disappointing that these models were not able to identify more definitive tells in the gameplay around bluffing, some interesting insights may be drawn from the variable importance plots of each model. Notably, the human players are more inclined to bet to try to “buy” a big pot. Intriguingly, Pluribus is not that sentimental about the pot and is more willing to let it go. A word of caution about over-interpreting this however - there is likely an inevitable feedback loop between bluffing and pot size. Namely, if a person folds instead of bluffing, naturally the pot will be smaller as well. Nonetheless, if there are any potentially predictive insights a player can make from the game about whether or not someone is bluffing are probably to be found while considering the size of the pot.

Finally, from a player’s perspective, it is interesting to see that, even after removing odds greater than 20% from the data, odds remains a major contributor for whether or not someone bluffs, including Pluribus. As seen in the boxplot above, it seems that the models confirm that bluffing while there is a path to victory is still the preferred strategy for players.

# Conclusion

Basic stats, correlation analysis, and association rules can certainly inform players on how bluffing is impacting the games they play and what strategy they should adopt. This data tells players who the big bluffers are at a table - namely, Gogo followed by Bill and then Pluribus. Correlation analysis above shows that bluffs are often when players are folding before the flop and it shows which players betting action lean toward or against bluffing. The association rules gives players even more specific clues about these scenarios when bluffing is occurring down to certain players battling with bets and the size of the POT being within a given range.

Modeling including SVM and Random Forest techniques provide prediction of actual strategic recommendations on bluffing. Output of the models show that with enough data fed in players can know with 75%-85% accuracy that someone is bluffing. By plotting these models players can actually see the scenarios they are losing to or winning from bluffing. As displayed in the SVM models, players could identify winning opportunities to bluff against Pluribus. Unfortunately, against these players in particular, trying to detect their bluff was more of a challenge. Against these particular players, it is best to just assume they are not bluffing and play to your own odds.

Ideally, follow up analysis would examine a deeper pool of games that would include more examples of bluffed games. Given the relatively small pool of bluffs in these series of hands, definitive insights may be limited by the amount of data. Additionally, it would be helpful to examine a wider range of players as well. If additional insights are to be gained through detecting bluffs or identifying successful bluffing strategies, it would be nice to know how well they generalize to other players in other settings.

Bluff analysis like this is useful to poker players for improving their own skills. They can see performance stats to know how much they should bluff and even, with the right algorithms, predict when a game could include a bluff. Indicators like their hand, flop, pot size, table size and who is the blind can inform players when bluffing is more common. Looking at the results, it is a real advantage players can have to hone in their knowledge of bluffing via data science. Imagine how powerful a future poker software can be that reads all the available data about the game and other players tendencies to suggest if they should fold or raise. With the advancement of machine learning and AI, the future for games of skill may need vast countering of AI algorithms and more thorough cheat detection.